NCC 3-387

7

283668

P-22

Performance Analysis Approximations for Parallel Processing of Concurrent Tasks

Erol Gelenbe
Zhen Liu

December 1, 1987

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 87.30

(NASA-CR-180476) PERFORMANCE ANALYSIS APPROXIMATIONS FOR PARALLEL PROCESSING OF CONCURRENT TASKS (Research Inst. for Advanced Computer Science) 22 p

N90-70740

Unclas
00/62 0280668

RIACS
Research Institute for Advanced Computer Science

Performance Analysis Approximations for Parallel Processing of Concurrent Tasks

Erol Gelenbe
Zhen Liu

December 1, 1987

Research Institute for Advanced Computer Science
NASA Ames Research Center

RIACS Technical Report 87.30

Abstract

This paper presents a method for obtaining approximate solutions to the performance analysis of parallel processing on multiprocessor systems composed of a finite number of homogeneous processors. In particular, we compute the expected response time of jobs consisting of concurrent tasks. We assume that each task requires an exponentially distributed amount of processing. The concurrency and precedence relations between tasks are described by directed acyclic graphs, referred to as task graphs or precedence graphs. Simulation results which we present indicate that the estimates provided by our method are lower bounds to the mean overall response time of tasks executed on the system.

1 Introduction

There has been increasing interest in performance analysis of concurrent programs in recent years. A concurrent program consists of a set of interdependent tasks with precedence constraints which are usually described by a task graph, also referred to as a precedence graph, whose nodes represent tasks and directed edges represent precedence relations between tasks. One of the most important

Work reported herein was supported in part by Cooperative Agreement NCC 2-387 between the National Aeronautics and Space Administration (NASA) and the Universities Space Research Association (USRA), and by a grant from Centre National d'Etudes des Telecommunications and the Centre National de la Recherche Scientifique (ATP mathematiques-Informatique).

performance measures of concurrent programs is the overall response time of a concurrent program running on multiprocessor systems.

When the durations of tasks are fixed and there are infinitely many processors, it is easy to determine the completion time of a concurrent program by using PERT diagrams. If resources are limited, research has been carried out in the area of task assignment and task scheduling. When task durations are allowed to be random, the performance analysis of concurrent programs becomes more difficult.

In order to make the analysis tractable, simple structures have been considered. Robinson [Rob 79], and Sahner and Trivedi [ST 87] obtain the overall execution time by restricting the graph structure to be *series-parallel.* Gelenbe et al. [GMSW 86] consider the same graph structure with random graph models and derive a closed form expression for the maximum speed-up which can be expected by executing such programs on a multiprocessor system.

Dodin [Dod 85] studies the general directed acyclic graph structure. By removing edges or spliting nodes, a directed acyclic graph can be transformed into a *series-parallel* graph. Thus lower bounds and upper bounds for overall completion time are obtained.

Gelenbe et al. [GNPT 86] model the task graph of a parallel computation by a random, directed, acyclic graph consisting of m vertices in which an arc from vertex i, $i = 1, 2, \cdots, m-1$ to vertex j, $j = i+1, \cdots, m$ exists with probability p. They derive an approximation for the expected processing time of a task graph of this type on an infinite number of processors under the assumption that each task requires an exponentially distributed amount of processing.

In this paper, we consider performance models of concurrent programs on finite resources. In particular, we consider the response time of concurrent programs executing on multiprocessor systems which consist of a finite number of homogeneous processors. The response time is defined as the delay between the arrival date of a job to the system and the date where every task of that job has completed its processing. At first, we assume that all jobs have the same task graph. This restriction is removed in Section 5. The task graphs under consideration have a general directed acyclic structure.

It is assumed that the processing demands of the tasks composing a concurrent program are independent with identical exponential distributions. The arrivals of jobs (i.e., concurrent programs) to the multiprocessor system are governed by a Poisson process. We also assume that tasks are assigned to processors just before their execution, and that there is no a priori knowledge of the processor upon which a task will be executed.

The model we consider is analytically intractable. In this paper, we obtain a computationally simple solution for the average delay of jobs in the system using an approximate technique based on product form networks.

In Section 2, we describe the problem in detail. In Section 3, the approximate solution is described, whereas in Section 4, approximate solution results are compared to simulation results. Section 5 provides an extension of our method to a multiprogramming environment.

2 Problem description

We assume that the multiprocessor system under consideration has M identical processors. A main memory is shared by all processors. It is assumed that this memory is of unlimited size so that it can store the address space of all programs which are present in the system. The workload consists of a set of structurally identical concurrent programs consisting of a fixed number of interdependent tasks. The precedence constraints between tasks are defined by an acyclic precedence graph in which each node represents a task.

We consider a stream of jobs (i.e., concurrent programs) arriving to the system according to Poisson process with parameter λ . All the jobs have the same precedence graph, denoted by G . Each task requires an exponentially distributed processing time with parameter μ . Processing times of different tasks are assumed to be independent. Let N be the maximum number of predecessors that tasks of graph G may have, $N \leq m - 1$, where m is the number of tasks in the job (or nodes in G).

Upon arrival, a job is immediately split into its constituent tasks so that tasks which have no predecessors are ready for immediate execution and enter directly into the processor queue waiting for service by one of the processors. The other tasks, which are not yet available for execution because of precedence constraints, have to wait in a buffer until their predecessors have been serviced.

We model this wait as follows. Tasks having i ($i = 1, 2, \cdots, N$) predecessors enter buffer i , and wait for their i predecessors' service completion. When the execution of a task is completed (by one of the processors), it leaves the system and may free some tasks waiting in one of the buffers. Once released by all its serviced predecessor tasks, a task leaves the buffer to join the server queue. An arriving job remains active in the system until all the tasks of the job have completed execution.

2.1 The general system model

An exact representation of the system could be the following. The state of the system is a collection of tasks which are either waiting for their predecessor tasks to be terminated, or waiting in queue, or being processed by one of the processors. For a task to be waiting in queue or to be processed, all of its predecessor tasks must be processed. Assuming FIFO service at the processors, the state must also represent the order of arrival of tasks, which is determined by the order in which the programs they belong to have arrived.

Each task will be denoted by a pair of integers (i, j) where i is the program number and j is the task number within the program. The state of the system will be represented by

$$S \equiv (W, Q)$$

where W and Q are defined as follows:

$$Q \equiv \begin{cases} (t_1, t_2, \cdots, t_{|Q|}), & t_i \ (1 \le i \le |Q|) \text{ is a task, and there} \\ & \text{are } |Q| \text{ tasks running or ready to run,} \\ 0, & \text{if there are no tasks in the system.} \end{cases}$$

The tasks t_l in Q are given in FIFO order with t_1 being at the head of the queue. Therefore the first M of these tasks will be actually running.

$$W \equiv \begin{cases} 0, & \text{if no tasks are waiting,} \\ (\omega_1, \omega_2, \cdots, \omega_{|W|}), & \text{if there are } |W| \text{ tasks waiting.} \end{cases}$$

where each ω_l is a pair

$$\omega_l = \{T_l, A_l\},$$

in which T_l is a task and A_l is the set of tasks which are the immediate predecessors of T_l . In fact, A_l will be unnecessary if all the programs running in the system have exactly the same precedence graph; in that particular case the task T_l uniquely identifies the set A_l for all programs. However if programs can have different task graphs, the set A_l will have to be specified.

The analysis of this general state-space model seems beyond the capability of analytical modelling. In order to illustrate the complexity of the model consider the state transition associated with the arrival of a program, from $S = (W, Q)$ to $S' = (W', Q')$:

$$W' \equiv \begin{cases} (\omega_1, \cdots, \omega_{|W|}, \omega'_{|W|+1}, \cdots, \omega'_{|W|+k}), & \text{if there are } k \text{ tasks with} \\ & \text{predecessors in the arriving program,} \\ 0, & \text{if } |W| = 0 \text{ and } k = 0 \text{ .} \end{cases}$$

$$Q' \equiv \begin{cases} (t_1, \cdots, t_{|Q|}, t'_{|Q|+1}, \cdots, t'_{|Q|+j}), & \text{if there are } j \text{ tasks without} \\ & \text{predecessors in the arriving program,} \\ 0, & \text{if } |Q| = 0 \text{ and } j = 0\,. \end{cases}$$

Obviously $\omega'_l = \{T'_l, A'_l\}$, so that the set of tasks of the arriving program is given by $\{T'_{|W|+1}, \cdots, T'_{|W|+k}, t'_{|Q|+1}, \cdots, t'_{|Q|+j}\}$. Furthermore the order in which the tasks t'_i which have no predecessors are placed in the processor queue Q' is of importance; we shall assume that they are placed in the order of increasing task number, so that the task with no predecessors and with the smallest task number is placed closest to the head of the queue. This will be the rule which will be used in general for placing tasks in the processor queue.

The representation of the other transitions related to task departures is even more complex. That is why in the rest of this paper we shall deal with a simplified model in which tasks and their predecessors will not be identified individually, and certain simplifying assumptions will be made about the state transitions.

2.2 The simplified model

In the simplified model, we shall consider generic tasks. The state of the system will be represented by the vector

$$\mathbf{k} = (k_0, k_1, \cdots, k_N)$$

where k_i represents the number of tasks in the system which have i $(1 \leq i \leq N)$ predecessor tasks. Recall that N is the maximum number of predecessors which a task may have. k_0 denotes the number of tasks which have no unfinished predecessors at the instant considered. This state representation is obviously much simpler than the general one. However it does not capture in a precis manner the structure of the task graphs. The state transitions form $\mathbf{k}$ to $\mathbf{k}'$ are given as follows for the simplified model.

- Arrival of a program or job represented by a sequence of task arrivals.

  Let λ be the arrival rate of programs to the system. We shall consider a Poisson arrival rate of tasks of rate λm , m being the total number of tasks per program. Thus we are modelling job arrivals by a flow of single task arrivals. Let m_i be the number (or average number for a random program graph) of tasks having i predecessors. Upon arrival of a task,

$$\mathbf{k}' = (k_0, \cdots, k_i + 1, \cdots, k_N), \qquad 0 \leq i \leq N$$

with probability chosen to be $p_i \equiv m_i/m$. Notice that $\sum_0^N m_i = m$. The rate of transition from state $\mathbf{k}$ to this particular $\mathbf{k}'$ is thus $\lambda m p_i$.

- Departure of a task from one of the wait buffers.

  We assume that transitions from state $\mathbf{k}$ to some state

$$\mathbf{k}' = (k_0 + 1, \cdots, k_i - 1, \cdots, k_N), \qquad 1 \le i \le N$$

  where $k_i > 0$, occur with rate $\mu_i(k_i)$. These represent the departure rate of a task from the wait buffer into the processor queue. $\mu_i(k_i)$ will be determined as a function of model parameters as given below.

- Departure of a task from the processor queue.

  This will lead into the state

$$\mathbf{k}' = (k_0 - 1, k_1, \cdots, k_N), \qquad k_0 > 0$$

  with rate

$$\mu_0(k_0) = \begin{cases} k_0\mu, & \text{if } 1 \le k_0 \le M \\ M\mu, & \text{if } k_0 > M \end{cases} \tag{1}$$

  Thus we are assuming that all individual tasks have independent exponentially distributed execution times of average value $1/\mu$. We may take $\mu = 1$ without loss of generality.

No other state transitions, except for those given above, are allowed to occur. Figure 1 illustrates the simplified model.

3 Delay analysis

The determination of the response time of a job described by a task graph G (i.e., the overall delay of graph G's tasks through the system), denoted by R_G, is of crucial importance in qualifying the performance of parallel processing. However hardly any theory exists to tackle such problems. The problem seems to be open in its general form.

For the sake of obtaining a tractable solution to the system, we restrict our attention to the simplified model, in which the following two simplifying assumptions have been made in Section 2.2.

> **A1)** Every buffer is replaced by a FIFO queue. The service time of queue i $(1 \le i \le N)$ is exponentially distributed with parameter μ_i defined in (3). When a customer of queue i is serviced, it joins the queue 0 (the multiprocessor queue).

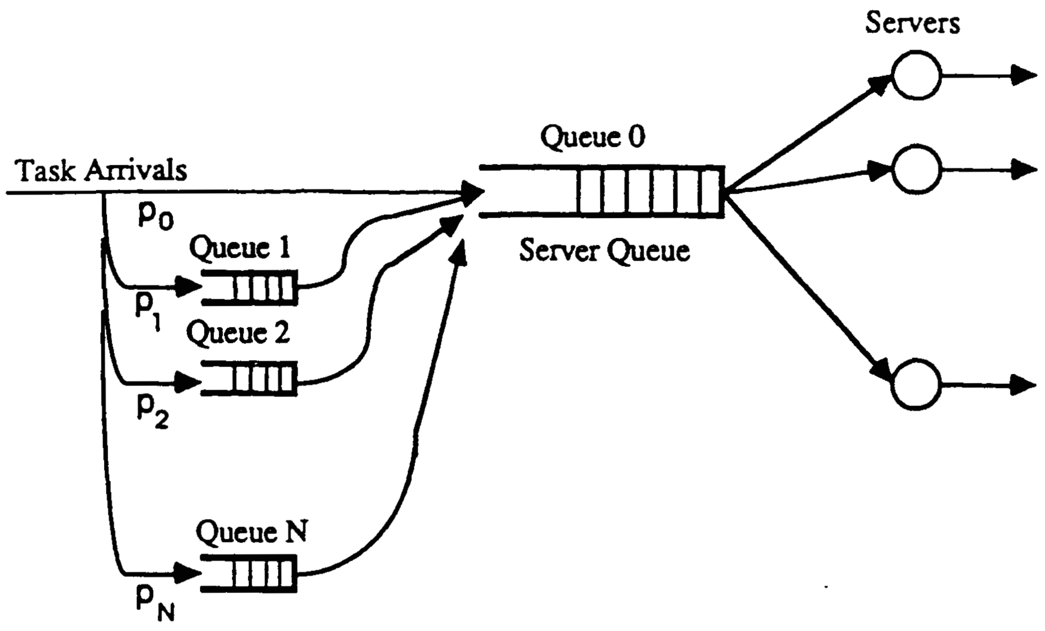


Figure 1: The Model Description

A2) The arrivals of tasks to the system constitute a Poisson process of parameter Λ, $\Lambda = m\lambda$. Let $T_1, T_2, \cdots, T_n, \cdots$ be the arriving tasks. We assume that tasks $T_{im+1}, T_{im+2}, \cdots, T_{im+m}$ form a precedence graph ($i = 1, 2, \cdots$). Upon arrival, a task becomes a customer of queue j ($0 \leq j \leq N$), denoted by C_j , with probability $p_j = m_j/m$.

With assumptions **A1** and **A2**, we obtain a product form network (c.f. [BCMP 75]) which provides the basis of our approach.

Before proceeding with the analysis, we introduce the following notation:

- $P(\mathbf{k})$: joint stationary queue length probability distribution,
- $P_i(k_i)$: stationary probability for queue i ($0 \leq i \leq N$) to have k_i ($n = 0, 1, 2, \cdots$) customers,
- Ek_i : mean number of tasks in queue i ($0 \leq i \leq N$),
- Ek : mean number of tasks in the system,
- R_i : mean response time of tasks in queue i ($0 \leq i \leq N$),
- R_t : mean response time of tasks in the system,
- Q : mean number of jobs in the system,

- R_G : mean graph response time of graph G, i.e., the overall delay of tasks of graph G through the system,

- $\overline{w_j}$: expected waiting time of j customers who simultaneously arrive at queue 0,

- q_i : service contribution to queue i $(0 \leq i \leq N)$ of a customer C_0 leaving queue 0.

The notion of ***service contribution*** is introduced in order to derive μ_i $(1 \leq i \leq N)$. Let G_v be the set of nodes of graph G, $Suc(v)$ the set of successors of v, $rank(v)$ the number of predecessors of node v. q_i is measured by the average number of successors having i predecessors.

$$q_i \overset{\text{def}}{=} \frac{1}{m}\Big(\sum_{v \in G_v} \sum_{v' \in Suc(v)} \mathbb{1}(rank(v') = i)\Big) \tag{2}$$

where $\mathbb{1}(x)$ is the characteristic function:

$$\mathbb{1}(x) = \begin{cases} 1, & \text{if } x \text{ is true} \\ 0, & \text{if } x \text{ is false} \end{cases}$$

Thus we may approximate the effect of the departure of a task from the system, on the set of waiting tasks, as the departure of (q_i/i) tasks on the average from the set of tasks in the i-th waiting queue. This leads to the approximation:

$$\mu_i = \gamma q_i / i \tag{3}$$

where γ is the effective departure rate of the multiprocessor service queue:

$$\gamma = \sum_{j=1}^{\infty} \mu_0(j) \cdot P_0(j) \tag{4}$$

Since our approximate model is an open product-form network, we have the following formula for the joint queue length probability ([BCMP 75]):

$$P(\mathbf{k}) = K \prod_{i=0}^{N} \prod_{n=1}^{k_i} \frac{\Lambda \cdot e_i}{\mu_i(n)} \tag{5}$$

where K is a normalising constant, μ_0 is expressed by (1), and by the nature of our model,

$$e_0 = 1 \tag{6}$$

$$e_i = p_i, \quad \text{if } 1 \leq i \leq N \tag{7}$$

and for $1 \leq i \leq N$

$$\mu_i(n) = \mu_i, \quad n = 1, 2, \cdots \tag{8}$$

We derive the marginal queue length probability distribution $P_i(k_i)$ ($i = 0, 1, \cdots, N$) in the following way.

Let $\mathbf{k}' = (k_0, \cdots, k_{i-1}, k_i + 1, k_{i+1}, \cdots, k_N)$. In using (5), $P(\mathbf{k}')$ can be expressed as

$$P(\mathbf{k}') = P(\mathbf{k}) \cdot (\Lambda e_i / \mu_i(k_i + 1)) \tag{9}$$

Since

$$P_i(k_i) = \sum_{k_0, \cdots, k_{i-1}, k_{i+1}, \cdots, k_N} P(k_0, \cdots, k_{i-1}, k_i, k_{i+1}, \cdots, k_N) \tag{10}$$

and

$$P_i(k_i + 1) = \sum_{k_0, \cdots, k_{i-1}, k_{i+1}, \cdots, k_N} P(k_0, \cdots, k_{i-1}, k_i + 1, k_{i+1}, \cdots, k_N) \tag{11}$$

Combining (9), (10), and (11), we obtain, for $n = 0, 1, 2, \cdots$

$$P_i(n+1) = P_i(n) \cdot \frac{\Lambda \cdot e_i}{\mu_i(n+1)} \tag{12}$$

Let

$$K_i \overset{\text{def}}{=} P_i(0) \tag{13}$$

then

$$P_i(n) = K_i \cdot \prod_{j=1}^{n} \frac{\Lambda \cdot e_i}{\mu_i(n+1)} \tag{14}$$

Summing over all $n = 0, 1, 2, \cdots$, we have

$$\sum_{n=0}^{\infty} P_i(n) = 1 \tag{15}$$

Hence we get the expression of K_i

$$K_i = (\sum_{n=0}^{\infty} \prod_{j=1}^{n} \frac{\Lambda \cdot e_i}{\mu_i(n+1)})^{-1} \tag{16}$$

Therefore, for $i = 0, 1, \cdots, N$ and $n = 0, 1, 2, \cdots$, (14) can be rewritten as

$$P_i(n) = (\sum_{n=0}^{\infty} \prod_{j=1}^{n} \frac{\Lambda \cdot e_i}{\mu_i(n+1)})^{-1} \cdot \prod_{j=1}^{n} \frac{\Lambda \cdot e_i}{\mu_i(n+1)} \tag{17}$$

For $i = 0, 1, \cdots, N$, the mean queue lengths are expressed as

$$Ek_i = \sum_{n=1}^{\infty} n \cdot P_i(n) \tag{18}$$

So the mean number of tasks in the system Ek is given by

$$Ek = \sum_{i=0}^{N} Ek_i \tag{19}$$

By Little's results, we obtain the formula for the average response time of a customer C_i $(i = 0, 1, \cdots, N)$

$$R_i = \frac{Ek_i}{e_i \Lambda} \tag{20}$$

and the average delay of a task R_t

$$R_t = \frac{\overline{N}}{\Lambda} \tag{21}$$

The above formulae provide performance measures at the task level. In particular, R_i $(i = 0, 1, \cdots, N)$ indicates the time that a task having i predecessors must expect to wait before becoming available for execution, and R_t indicates the expected delay of a task passing through the system.

We now compute R_G, the average response time of jobs. First of all, we define the concept of *level*.

The *level* of a node v of a directed acyclic graph, denoted by $L(v)$, is defined as follows:

1. $L(v) \overset{\text{def}}{=} 1$, if $v \in G_v$ and v has no predecessor,
2. $L(v) = (\max_{p \in Pre(v)} L(p)) + 1$, where $Pre(v)$ denotes the set of immediate predecessors of node v.

The *level* of a task is considered as the *level* of the corresponding node in the task graph. The *level* of a graph, denoted by L_G, is given by

$$L_G = \max_{v \in G_v} L(v)$$

In order to estimate the mean response time of jobs, we neglect the fact that task processing times will vary in general, and we assume that they are constant. Therefore we assume that tasks of a same job are serviced by the processors level by level, and a task of level i can be serviced if and only if all tasks of level $i-1$ belonging to the same job have completed execution. Furthermore we assume that

A3) Tasks of the same level become simultaneously available for execution.

Let $t_1, t_2, \cdots, t_u$ be a group of tasks concurrently arriving at the multiprocessor service queue. Without loss of generality, we assume that their service order is $t_1 \prec t_2 \prec \cdots \prec t_u$. Let $w_k(n)$ denote the mean waiting time of task t_k $(1 \leq k \leq u)$ before service if there are n $(n = 0, 1, 2, \cdots)$ tasks in the queue at the instant of the arrival of the group. Thus we have the formula

$$w_k(n) = \begin{cases} 0 & \text{if } n \leq M - k \\ (n + k - M)\mu^{-1}/M & \text{if } n \geq M - k + 1 \end{cases} \tag{22}$$

Therefore, for $j = 1, 2, \cdots$, the expected waiting time of j concurrent tasks in the server queue can be expressed as

$$\overline{w_j} = \sum_{n=0}^{\infty} P_0(n) \cdot w_j(n) \tag{23}$$

or

$$\overline{w_j} = \sum_{n=max(M-k+1,0)}^{\infty} P_0(n) \cdot (n + k - M) \cdot \mu^{-1}/M \tag{24}$$

Let $b(l)$ denote the number of tasks of level l $(1 \leq l \leq L_G)$. According to assumption **A3**, tasks are serviced level by level, and tasks of the same level concurrently arrive at the multiprocessor queue. Therefore the job response time of graph G, R_G, may be expressed as the sum of the completion time of each level:

$$R_G = \sum_{l=1}^{L_G} (\frac{1}{\mu} + \overline{w}_{b(l)}) \tag{25}$$

And finally, by Little's result, we get the average number of jobs in the system Q,

$$Q = \lambda \cdot R_G \tag{26}$$

Notice that the synchronisation delay of tasks induced by the precedence graph is taken into account in computing the mean number of tasks and the task response time. In computing the mean delay of a job passing through the system, however, the synchronisation delay of tasks is ommited since we neglect the variability of the service times of the tasks.

4 Comparison of approximations with simulation results

In this section we compare our approximations of expected job response time against results obtained from simulations. The task graph examples are selected from various parallel processing application areas such as numerical analysis, image processing, etc. Their structures are presented in the Appendix. Tables 1-5 report results of comparisons of approximations with simulations. Figure 2 illustrates the response times of graph 3 under different system load conditions. The simulations have been carried out with the QNAP2 package. 95% confidence intervals have been calculated with the regeneration method. The system load used in Tables 1 to 5 is defined by

$$\rho = \frac{m\lambda}{M\mu} \tag{27}$$

Table 1. Comparison of job response time for graph 1.
($\lambda = 0.06$, $\mu = 1$)

number of processors	system load	approximate solution	simulation estimate	relative error %
1	0.720	32.57	46.84±2.45	30.3
2	0.360	11.19	12.21±0.22	8.3
3	0.485	8.71	9.27±0.09	6.1
5	0.240	8.02	8.39±0.06	4.4
10	0.072	8.00	8.30±0.06	3.6

Table 2. Comparison of job response time for graph 2.
($\lambda = 0.06$, $\mu = 1$)

number of processors	system load	approximate solution	simulation estimate	relative error %
1	0.960	184.00	241.3±36.23	23.7
2	0.480	14.13	17.69±0.44	20.2
3	0.320	10.85	12.48±0.18	13.1
5	0.383	7.35	10.49±0.07	30.0
10	0.096	7.00	10.21±0.06	31.4

Table 3. Comparison of job response time for graph 3.
($\lambda = 0.06$, $\mu = 1$)

number of processors	system load	approximate solution	simulation estimate	relative error %
1	0.960	256.00	260.2±37.61	1.6
2	0.480	14.94	18.69±0.44	20.1
3	0.320	10.92	13.78±0.16	20.8
5	0.192	10.03	12.59±0.08	20.3
10	0.096	10.00	12.49±0.07	19.9

Table 4. Comparison of job response time for graph 4.
($\lambda = 0.06$, $\mu = 1$)

number of processors	system load	approximate solution	simulation estimate	relative error %
1	0.900	78.00	134.3±14.89	41.9
2	0.450	12.78	16.10±0.45	20.6
3	0.300	9.79	10.10±0.12	3.0
5	0.180	7.21	8.21±0.06	12.2
10	0.090	7.00	7.89±0.05	11.3

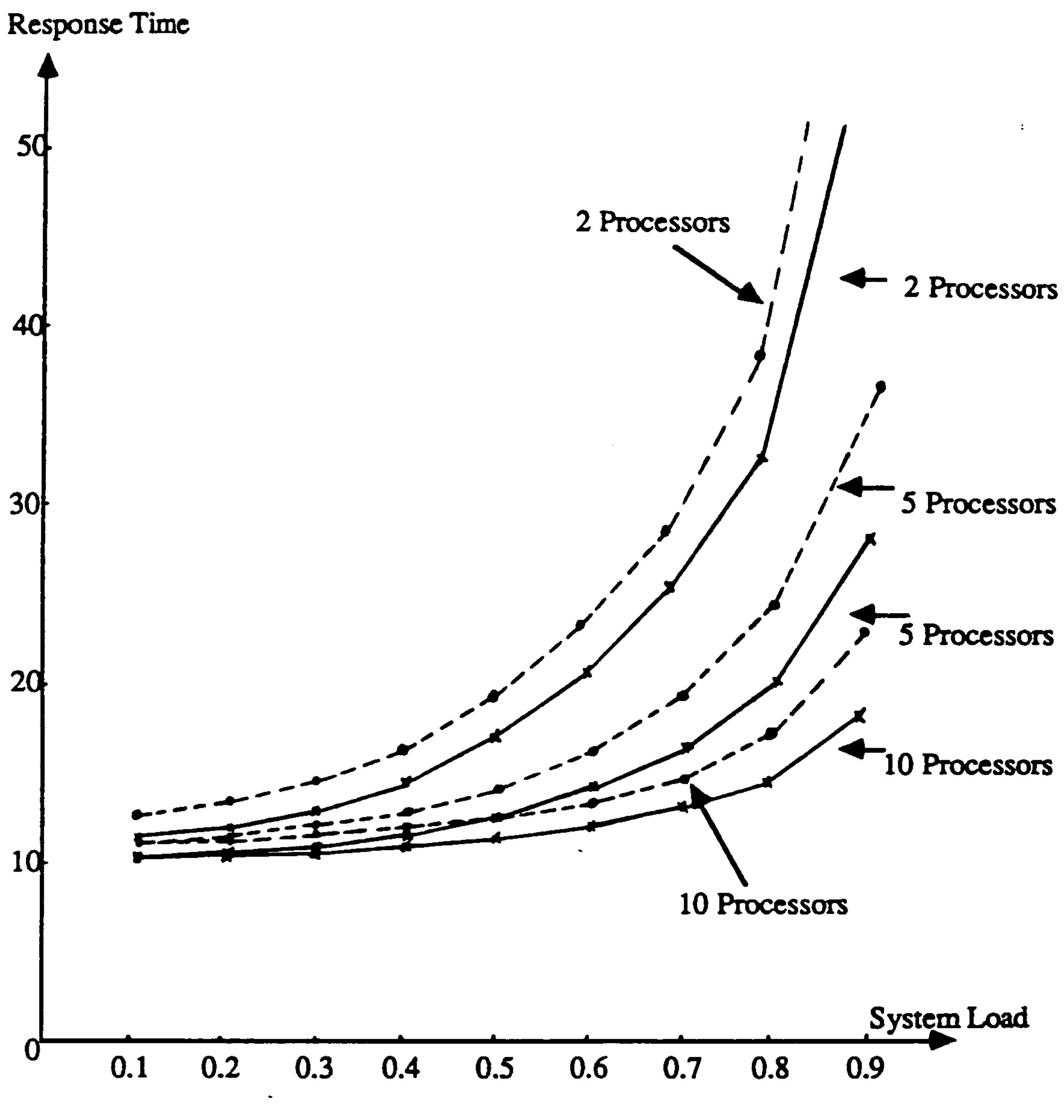
Table 5. Comparison of job response time for graph 5.
($\lambda = 0.06$, $\mu = 1$)

number of processors	system load	approximate solution	simulation estimate	relative error %
1	0.960	208.00	296.1±44.88	29.8
2	0.480	14.75	18.32±0.52	19.5
3	0.320	11.06	11.79±0.19	6.2
5	0.192	8.25	9.36±0.06	11.9
10	0.096	8.00	8.98±0.05	11.0

Our approximations compare favorably with the simulation results. The relative errors are mostly within 10 or 20%, and in the worst case within 40% of the simulation results. The fact that our approximation underestimates the overall delay can be explained by the three simplifying assumptions. Indeed:

1. Bulk arrival queueing systems behave generally worse than single arrival systems under the same load.

2. Dependent services (in our model, services of queue 1 to N) generally imply a larger variance of service time, so that the waiting time is greater.



------ : Simulation Results

——— : Approximations

Figure 2: Response Times of Graph 3 Under Different System Load Conditions

3. Synchronization of tasks introduces additional delays to task execution times.

These intuitive arguments indicate that our approximation provides lower bounds to expected response time of jobs.

5 The extension of the method to multiprogramming systems

The extension of our approximate solution to the performance analysis of parallel processing in multiprogramming environments is immediate. In this section, we shall only discuss it briefly. The notation, if not redefined, is taken from Section 3.

In a multiprogramming system, different types of jobs are allowed to be executed simultaneously. Let $G_1, G_2, \cdots, G_H$ $(H \geq 1)$ denote task graphs of jobs of classes $1, 2, \cdots, H$, respectively. We assume that arrivals of jobs of classe h $(h = 1, 2, \cdots, H)$, are generated by independent Poisson processes with parameter λ_h. Jobs of class h have m_h tasks which are divided into N_h $(N_h \leq m_h - 1)$ groups according to their respective number of predecessors. Let

$$N = \max_{1 \leq h \leq H} N_h \tag{28}$$

Every task (whichever class or job it belongs to) is assumed to require an exponentially distributed amount of processing with parameter μ.

By replacing bulk arrivals with Poisson arrivals, and replacing waiting buffers with independent exponential service queues, we obtain a new simplified model which is similar to the one in Section 3. We thus consider a stream of tasks $t_1, t_2, \cdots, t_n, \cdots$ arriving to the system, where t_n $(n \geq 1)$ belongs to a job of class j with probability $m_j / \sum_{h=1}^{H} m_h$. The arrivals are represented by a Poisson process with parameter Λ,

$$\Lambda = \sum_{h=1}^{H} m_h \cdot \lambda_h \tag{29}$$

Upon arrival, a task becomes a customer of queue i $(0 \leq i \leq N)$ with probability p_i,

$$p_i = \frac{\sum_{h=1}^{H} m_{hi}}{\sum_{h=1}^{H} m_h} \tag{30}$$

where

$$m_{hi} = \textit{number of tasks in } G_h \textit{ having } i \textit{ predecessors.} \tag{31}$$

which satisfies

$$\sum_{i=0}^{N_h} m_{hi} = m_h, \quad h = 1, 2, \cdots, H \tag{32}$$

The concept *service contribution* is represented by

$$q_i = \frac{1}{\sum_{h=1}^{H} m_h} \left(\sum_{h=1}^{H} \sum_{v \in G_v^h} \sum_{v' \in Suc^h(v)} 1(rank(v') = i) \right) \tag{33}$$

where G_v^h denotes the set of nodes of graph G_h, $Suc^h(v)$ denotes the set of successors of v.

Formulae (5)-(8), (17)-(21) remain valid for the queue length probabilities, mean queue lengths, mean number of tasks in the system, and average delays of tasks. Furthermore, the mean number of tasks which belong to jobs of class j ($j = 1, 2, \cdots, H$) in queue i ($i = 0, 1, \cdots, N$), denoted by Ek_{ji}, can be expressed as

$$Ek_{ji} = \frac{m_{ji}\lambda_j}{\sum_{h=1}^{H} m_{hi}\lambda_h} Ek_i \tag{34}$$

where Ek_i is expressed by (18), and the mean number of tasks which belong to jobs of class j ($j = 1, 2, \cdots, H$) in the system, denoted by Ek^j, can be expressed as

$$Ek^j = \frac{m_j\lambda_j}{\sum_{h=1}^{H} m_h\lambda_h} Ek \tag{35}$$

where Ek is expressed by (19).

The idea for computing the expected response time of a job of class h remains the same. Maintaining the assumption **AS**, we can rewrite (25) as follows

$$R_{G_h} = \sum_{l=1}^{L_{G_h}} \left(\frac{1}{\mu} + \overline{w}_{b_h(l)} \right) \tag{36}$$

where R_{G_h} denotes the expected response time of a job of class h, L_{G_h} denotes the level of graph G_h, $b_h(l)$ denotes the number of tasks in graph G_h at level l ($1 \le l \le L_{G_h}$), and $\overline{w_j}$ ($j = 1, 2, \cdots$) is given by (24).

Similar to (26), the average number of active jobs of class h is expressed as:

$$Q_h = \lambda_h \cdot R_{G_h} \tag{37}$$

and the average number of jobs in the system

$$Q = \sum_{h=1}^{H} Q_h \tag{38}$$

As above described, we obtain a computationally simple estimates for performance analysis for parallel processing in multiprogramming environments.

6 Conclusions

Parallelism is often limited by precedence relations within programs and finite resources in parallel processing systems. Performance predictions for concurrent programs on multiprocessor systems are of crucial importance for both software and hardware designers. Nevertheless hardly any theory is available for analysing the performance of concurrent programs which have general directed acyclic graph structures and execute on multiprocessor systems with a limited number of processors.

In this paper, we have considered the expected response time of concurrent programs executed on multiprocessor systems with a finite number of processors. We have introduced an approximate model to estimate the performance measures of parallel processing. The method has been extended to multiprogramming systems.

The multiprocessor system under consideration consists of a finite number of homogeneous processors which share a central memory. Arriving jobs (i.e., concurrent programs) consist of tasks with identical exponential execution time distribution. The concurrency and precedence relationship between tasks are defined by directed acyclic graphs of general structure. The arrivals of jobs are assumed to be generated by a Poisson process.

Because of the concurrency, the queueing network resulting from such a system does not have a product-form solution. An approximate solution method has been described which models the system by a set of waiting queues for tasks whose predecessors have not yet terminated their execution, and a central server with variable service rate. Some simplifying assumptions have been made to derive the estimates of the response time of jobs. The accuracy of the approximation has been reported through comparisons against simulation results. Some arguments have been provided to indicate that our approximations are lower bounds to the expected response time of jobs.

Future research topics include investigating approximate solutions to performance estimates of concurrent programs consisting of tasks with different service distributions, and approximate solutions to the case where task assignment strategies are static, which means that each task has a previously specified processor destination.

References

[BCMP 75]: Baskell,F., Chandy,K.M., Muntz,R.R. and Palacios,F.G. *Open, Closed, and Mixed Networks of Queues with Different Classes of Customers.* J.ACM Vol.22, pp.248-260 (1975)

[Dod 85]: Dodin,B. *Bounding the Project Completion Time Distribution in PERT Networks.* Operations Research, Vol.33, No.4, pp.862-881 (July-August 1985)

[GMSW 86]: Gelenbe,E., Montagne,E., Suros,E. and Woodside,C.M. *A Performance Model of Block Structured Parallel Programs.* ISEM research report n^o46, Université de Paris-Sud, France (1986)

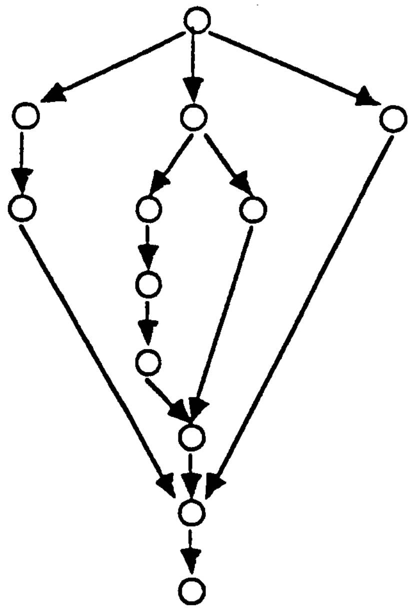
[GNPT 86]: Gelenbe,E., Nelson,R., Philips,T. and Tantawi,A. *The Asymptotic Processing Time for a Model of Parallel Computation.* Proceedings of National Computer Conference, Las Vegas, 1986.

[Rob 79]: Robinson,J.T. *Some Analysis Techniques for Asynchronous Multiprocessor Algorithms.* IEEE Trans. on Software Engineering, Vol.SE-5, No., pp.24-31 (Jan. 1979)

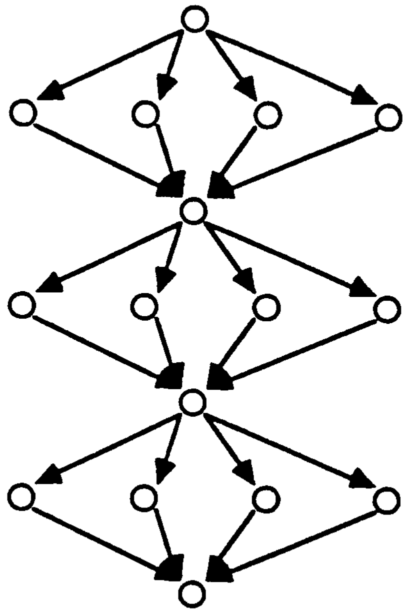
[ST 87]: Sahner,R.A. and Trivedi,K.S. *Performance and Reliability Analysis Using Directed Acyclic Graphs.* To appear in IEEE Trans. on Software Engineering.

7 Appendix: Task graph examples

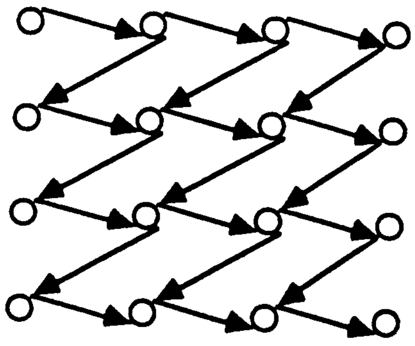
Graph.1



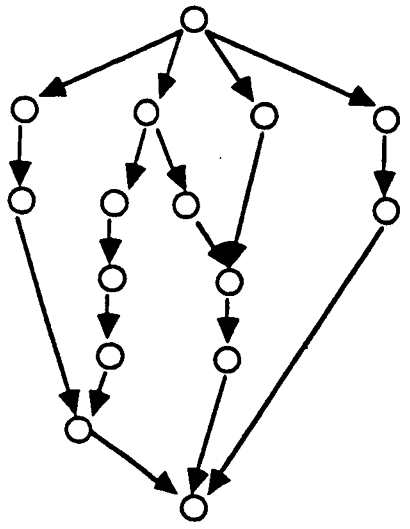
Graph.2



Graph.3



Graph.4



Graph.5